(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 April 2002 (11.04.2002)

PCT

(10) International Publication Number
WO 02/29573 A2

(51) International Patent Classification[7]: G06F 11/14

(21) International Application Number: PCT/US01/25763

(22) International Filing Date: 17 August 2001 (17.08.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/642,061 18 August 2000 (18.08.2000) US

(71) Applicant: **NETWORK APPLIANCE, INC.** [US/US]; 495 East Java Drive, Sunnyvale, CA 94089 (US).

(72) Inventors: **LEWIS, Blake**; 2316 Greer Road, Palo Alto, CA 94303 (US). **EDWARDS, John, K.**; 1173 Crandano Court, Sunnyvale, CA 94087-2076 (US). **VISWANATHAN, Srinivasan**; 751 Saltillo Place, Fremont, CA 94536 (US).

(74) Agent: **SWERNOFSKY, Steven, A.**; Swernofsky Law Group, P.O. Box 390013, Moutain View, CA 94039-0013 (US).

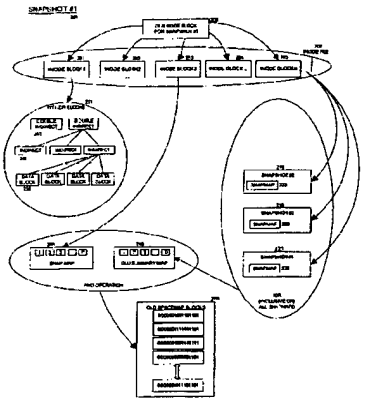
(84) Designated States *(regional)*: European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR).

Declaration under Rule 4.17:
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii)) for all designations*

Published:
— *without international search report and to be republished upon receipt of that report*

[Continued on next page]

(54) Title: INSTANT SNAPSHOT



(57) Abstract: The invention provides an improved method and apparatus for creating a snapshot of a file system. In a first aspect of the invention, a "copy-on-write" mechanism is used. An effective snapshot mechanism must be efficient both in its use of storage space and in the time needed to create it because file systems are often large. The snapshot uses the same blocks as the active file system until the active file system is modified. Whenever a modification occurs, the modified data is copied to a new block and the old data is saved (henceforth called "copy-on-write". In this way, the snapshot only uses space where it differs from the active file system, and the amount of work required to create the snapshot is small. In a second aspect of the invention, a record of which blocks are being used by the snapshot is included in the snapshot itself, allowing effectively instantaneous snapshot creation and deletion. In a third aspect of the invention, the state of the active file system is described by a set of metafiles; in particular, a bitmap henceforth the "active map") describes which blocks are free and which are in use. The inode file describes which blocks are used by each file, including the metafiles. The inode file itself is described by a special root inode, also known as the "fsinfo block". The system begins creating a new snapshot by making a copy of the root inode. This copy of the root inode becomes the root of the snapshot. The root inode captures all required states for creating the snapshot such as the location of all files and directories in the file sysptem, it. During subsequent updates of the active file system, the system consults the bitmap included in the snapshot (the "snapmap") to determine whether a block is free for reuse or belongs to the snapshot. This mechanism allows the active file system to keep track of which blocks each snapshot uses without recording any additional bookkeeping information in the file system. In a fourth aspect of the invention, a snapshot can also be deleted instantaneously simply by discarding its root inode. Further bookkeeping is not required, because the snapshot includes its own description. In a fifth aspect of the invention, the performance overhead associated with the search for free

[Continued on next page]

WO 02/29573 A2

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

blocks is reduced by the inclusion of a summary file. The summary file identifies blocks that are used by at least one snapshot; it is logical OR of all the snapmap files. The write location code decides whether a block is free by examining the active map and the summary file. The active map indicates whether the block is currently in use in the active file system. The summary file indicates whether the block is used by any snashot. In a sixth aspect of the invention, the summary file is updated in the background after the creation or deletion of the snapshot. This occurs concurrently with other file system operations. Two bits are stored in the file system "fsinfo block" for each snapshot. These two bits indicate whether the summary file needs to be updated using the snapshot's snapmap information as a consequence of its creation or deletion. When a block is freed in the active file system, the corresponding block of the summary file is updated with the snapmap from the most recently created snapshot, if this has not already been done. An in-core bit map records the completed updates to avoid repeating them unnecessarily. This ensures that the combination of the active bitmap and the summary file will consistently identify all blocks that are currently in use. Additionally, the summary file is updated to reflect the effect of any recent snapshot deletions when freeing a block in the active file system. This allows reuse of blocks that are not entirely free. After updating the summary file following a snapshot creation or deletion, the corresponding bit in the fsinfo block is adjusted. In a seventh a

INSTANT SNAPSHOT

Background of the Invention

1. *Field of Invention*

This invention relates to data storage systems.

2. *Related Art*

Snapshots of a file system capture the contents of the files and directories in a file system at a particular point in time. Such snapshots have several uses. They allow the users of the file system to recover earlier versions of a file following an unintended deletion or modification. The contents of the snapshot can be copied to another storage device or medium to provide a backup copy of the file system; a snapshot can also be copied to another file server and used as a replica. The WAFL (Write Anywhere File Layout) file system includes a copy-on-write snapshot mechanism. Snapshot block ownership in WAFL has been recorded by updating the block's entry in a blockmap file, which is a bitmap indicating which blocks are in-use and which are free for use.

One problem with the prior art of creating snapshots is that the requirement for additional file system metadata in the active file system to keep track of which blocks snapshots occupy. These methods are inefficient both in their use of storage space and in the time needed to create the snapshots.

A second problem with earlier snapshot implementations, was the time consuming steps of writing out a description of the snapshot state on creation and removing it on deletion.

A third problem with earlier copy-on-write mechanisms, was the required steps consumed a considerable amount of time and file system space. For example, some systems, such as those supplied with DCE/DFS, include a copy-on-write mechanism for creating snapshots (called "clones"). The copy-on-write mechanism was used to record which blocks each clone occupied. Such systems require a new copy of the inode file and

the indirect blocks for all files and directories are created when updating all of the original inodes.

Accordingly, it would be advantageous to provide an improved technique for more quickly and efficiently capturing the contents of the files and directories in the file system at a particular point in time. This is achieved in an embodiment of the invention that is not subject to the drawbacks of the related art.

Summary of the Invention

The invention provides an improved method and apparatus for creating a snapshot of a file system.

In a first aspect of the invention, the file system uses the fact that each snapshot includes a representation of the complete active file system as it was at the time the snapshot was made, including the blockmap of disk blocks indicating which ones are free and which ones are in use (herein called the "active map"). Because a record of which blocks are being used by the snapshot is included in the snapshot itself, the file system can create and delete snapshots very quickly. The file system uses those recorded blockmaps (herein called "snapmaps") as a source of information to determine which blocks cannot be reused because those blocks are being used by one or more snapshots.

In a second aspect of the invention, the file system uses that fact that it need only maintain a more limited blockmap of those disk blocks in use by the active file system, and a summary map of those disk blocks in use by one or more snapshots. The summary map can be computed from the snapmaps as the logical inclusive-OR of all the snapmaps. Because the file system need not maintain multiple bits of in-use/free data for each block, it uses the active map in conjunction with the summary map to determine whether blocks are in-use or free.

In a third aspect of the invention, the file system makes use of the fact that the summary map need not be updated every time a block is allocated or freed. Accordingly, the file system updates the summary map only (1) when a snapshot is deleted, and then only in a background operation, (2) on demand for areas for which write allocation is about to be

performed, and (3) periodically in a background operation for selected portions of the summary map. These background operations are preferably performed concurrently with other file system operations.

Information is stored in a persistent storage medium accessible by the file system, to provide for resumption of operation following a reboot operation. For example, in a preferred embodiment, relevant information is stored in the file system "fsinfo block" for each snapshot, to indicate whether the summary file needs to be updated using that snapshot's snapmap information as a consequence of its creation or deletion. When a block is freed in the active file system, the corresponding block of the summary file is updated with the snapmap from the most recently created snapshot, if this has not already been done. An in-core bit map records the completed updates to avoid repeating them unnecessarily. This ensures that the combination of the active bitmap and the summary file will consistently identify all blocks that are currently in use. Additionally, the summary file is updated to reflect the effect of any recent snapshot deletions when freeing a block in the active file system. This allows reuse of blocks that are now entirely free. After updating the summary file following a snapshot creation or deletion, the corresponding bit in the fsinfo block is adjusted.

In a fourth aspect of the invention, the algorithm for deleting a snapshot involves examining the snapmaps of the deleted snapshot and the snapmaps of the next oldest and next youngest snapshot. A block that was used by the deleted snapshot but is not used by its neighbors can be marked free in the summary file, as no remaining snapshot is using it. However, these freed blocks cannot be reused immediately, as the snapmap of the deleted snapshot must be preserved until summary updating is complete. During a snapdelete free blocks are found by using the logical OR of the active bitmap, the summary file, and the snapmaps of all snapshots for which post-deletion updating is in progress. In other words, the snapmap of the deleted snapshot protects the snapshot from reuse until it is no longer needed for updating.

In the preferred embodiment, the invention is operative on WAFL file system. However, it is still possible for the invention to be applied to any computer data storage system such as a database system or a store and forward system such as cache or RAM if the data is kept for a limited period of time.

Brief Description of the Drawings

Figure 1 shows a block diagram of a system for an instant snapshot.

Figure 2 shows a block diagram of an instant snapshot.

Figure 3 shows a flow diagram of a method for creating a snapshot.

Figure 4 shows a flow diagram of a method for updating a summary map.

Figure 5 shows a block diagram of copy-on-write maintenance of the active map.

Detailed Description of the Preferred Embodiment

In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. However, those skilled in the art would recognize, after perusal of this application, that embodiments of the invention might be implemented using a variety of other techniques without undue experimentation or further invention, and that such other techniques would be within the scope and spirit of the invention.

Lexicography

As used herein, use of the following terms refer or relate to aspects of the invention as described below. The general meaning of these terms is intended to be illustory and in no way limiting.

- fsinfo (File System Information Block) — In general, the phrase "file system information block" refers to one or more copies of a block known as the "fsinfo block". These blocks are located at fixed locations on the disks. The fsinfo block

includes data about the volume including the size of the volume, volume level options, language and more.

- WAFL (Write Anywhere File Layout) — In general, the term "WAFL" refers to a high level structure for a file system. Pointers are used for locating data. All the data is included in files. These files can be written anywhere on the disk in chunks of file blocks placed in data storage blocks.

- Consistency Point (CP) — In general, the term "CP" refers to a time that a file system reaches a consistent state. When this state is reached, all the files have been written to all the blocks and are safely on disk and the one or more copies of redundant fsinfo blocks get written out. If the system crashes before the fsinfo blocks go out, all other changes are lost and the system reverts back to the last CP. The file system advances atomically from one CP to the next.

- Consistent State — In general, the phrase "consistent state" refers to the system configuration of files in blocks after the CP is reached.

- Active file system — In general, the phrase "active file system" refers to the current file system arrived at with the most recent CP. In the preferred embodiment, the active file system includes the active map, the summary map and points to all snapshots and other data storage blocks through a hierarchy of inodes, indirect data storage blocks and more.

- Active map — In general, the phrase "active map" refers to a to a file including a bitmap associated with the in-use or free status of blocks of the active file system.

- Snapshot — In general, the term "snapshot" refers to a copy of the file system. The snapshot diverges from the active file system over time as the active file system is modified. A snapshot can be used to return the file system to a particular CP (consistency point).

- Snapmap — In general, the term "snapmap" refers to a file including a bitmap associated with the vacancy of blocks of a snapshot. The active map diverges from a snapmap over time as the blocks used by the active file system change during consistency points.

- Summary map — In general, the term "summary map" refers to a file including an IOR (inclusive OR) bitmap of all the snapmaps.

- Space map — In general, the term "space map" refers to a file including an array of numbers which describe the number of storage blocks used in an allocation area.

- Blockmap — In general, the term "blockmap" refers to a map describing the status of the blocks in the file system.

- Snapdelete — In general, the term "snapdelete" refers to an operation that removes a particular snapshot from the file system. This command can allow a storage block to be freed for reallocation provided no other snapshot or the active file system uses the storage block.

- Snapcreate — In general, the term "snapcreate" refers to the operation of retaining a consistency point and preserving it as a snapshot.

As described herein, the scope and spirit of the invention is not limited to any of the definitions or specific examples shown therein, but is intended to include the most general concepts embodied by these and other terms.

System Elements

Figure 1 shows a block diagram of a system for an instant snapshot.

The root block 100 includes the inode of the inode file 105 plus other information regarding the active file system 110, the active map 115, previous active file

systems known as snapshots 120, 125, 130 and 135 and their respective snapmaps 140, 145, 150 and 155.

The active map 115 of the active file system 110 is a bitmap associated with the in-use or free status of blocks for the active file system 110. The respective snapmaps 140, 145, 150 and 155 are active maps can be associated with particular snapshots 120, 125, 130 and 135 and an inclusive OR summary map 160 of the snapmaps 140, 145, 150 and 155. Also shown are other blocks 115 including double indirect blocks 130 and 132, indirect blocks 165, 166 and 167 and data blocks 170, 171, 172 and 173. Finally, Figure 1 shows the spacemap 180 including a collection of spacemap blocks of numbers 181, 182, 183, 184 and 190.

The root block 100 includes a collection of pointers that are written to the file system when the system has reached a new CP (consistency point). The pointers are aimed at a set of indirect (or triple indirect, or double indirect) inode blocks (not shown) or directly to the inode file 105 consisting of a set of blocks known as inode blocks 191, 192, 193, 194 and 195.

The number of total blocks determines the number of indirect layers of blocks in the file system. The root block 100 includes a standard quantity of data, such as 128 bytes. 64 of these 128 bytes describe file size and other properties; the remaining 64 bytes are a collection of pointers to the inode blocks 191, 192, 193, 194 and 195 in the inode file 105. Each pointer in the preferred embodiment is made of 4 bytes. Thus, there are approximately 16 pointer entries in the root block 100 aimed at 16 corresponding inode blocks of the inode file 105 each including 4K bytes. If there are more than 16 inode blocks, indirect inode blocks are used.

In a preferred embodiment, file blocks are 4096 bytes and inodes are 128 bytes. It follows that each block of the inode file contains 32 (i.e. 4,096/128) separate inodes that point to other blocks 115 in the active file system.

Inode block 193 in the inode file 105 points to a set of blocks (1, 2, 3, ..., P) called the active map 115. Each block in the active map 115 is a bitmap where each bit corresponds to a block in the entire volume. A "1" in a particular position in the bitmap

correlates with a particular allocated block in the active file system 110. Conversely, a "0" correlates to the particular block being unused by the active file system 110. Since each block in the active map 115 can describe up to 32K blocks or 128 MB, 8 blocks are required per GB, 8K blocks per TB.

Another inode block in the inode file 105 is inode block N 212. This block includes a set of pointers to a collection of snapshots 120, 125, 130 and 135 of the volume. Each snapshot includes all the information of a root block and is equivalent to an older root block from a previous active file system. The snapshot 120 may be created at any past CP. Regardless when the snapshot is created, the snapshot is an exact copy of the active file system at that time. The newest snapshot 120 includes a collection of pointers that are aimed directly or indirectly to the same inode file 105 as the root block 100 of the active file system 110.

As the active file system 110 changes (generally from writing files, deleting files, changing attributes of files, renaming file, modifying their contents and related activities), the active file system and snapshot will diverge over time. Given the slow rate of divergence of an active file system from a snapshot, any two snapshots will share many of the same blocks. The newest snapshot 120 is associated with snapmap 140. Snapmap 140 is a bit map that is initially identical to the active map 115. The older snapshots 125, 130 and 194 have a corresponding collection of snapmaps 145, 150 and 155. Like the active map 115, these snapmaps 145, 150 and 155 include a set of blocks including bitmaps that correspond to allocated and free blocks for the particular CP when the particular snapmaps 145, 150 and 155 were created. Any active file system may have a structure that includes pointers to one or more snapshots. Snapshots are identical to the active file system when they are created. It follows that snapshots contain pointers to older snapshots. There can be a large number of previous snapshots in any active file system or snapshot. In the event that there are no snapshot, there will be no pointers in the active file system.

Blocks not used in the active file system 110 are not necessarily available for allocation or reallocation because the blocks may be used by snapshots. Blocks used by snapshots are freed by removing a snapshot using the snapdelete command. When a snapshot is deleted any block used only by that snapshot and not by other snapshots nor by the active file system becomes free for reuse by WAFL. If no other snapshot or active files

uses the block, then the block can be freed, and then written over during the next copy-on-write operation by WAFL.

The system can relatively efficiently determine whether a block can be removed using the "nearest neighbor rule". If the previous and next snapshot do not allocate a particular block in their respective snapmaps, then the block can be freed for reuse by WAFL. For WAFL to find free space to write new data or metadata, it could search the active map 115 and the snapmaps (140, 145, 150 and 155) of the snapshots (120, 125, 130 and 135) to find blocks that are totally unused. This would be very inefficient; thus it is preferable to use the active map and the summary map as described below

A summary map 160 is created by using an IOR (inclusive OR) operation 139 on the snapmaps 140, 145, 150 and 155. Like the active map 115 and the snapmaps 140, 145, 150 and 155, the summary map 160 is a file whose data blocks (1, 2, 3, ...Q) contained a bit map. Each bit in each block of the summary map. describes the allocation status of one block in the system with "1" being allocated and "0" being free. The summary map 160 describes the allocated and free blocks of the entire volume from all the snapshots 120, 125, 130 and 135 combined. The use of the summary file 160 is to avoid overwriting blocks in use by snapshots.

An IOR operation on sets of blocks (such as 1,024 blocks) of the active map 115 and the summary map 160 produces a spacemap 180. Unlike the active map 115 and the summary map 160, which are a set of blocks containing bitmaps, the spacemap 180 is a set of blocks including 181, 182, 183, 184, and 190 containing arrays of binary numbers. The binary numbers in the array represent the addition of all the vacant blocks in a region containing a fixed number of blocks, such as 1,024 blocks. The array of binary numbers in the single spacemap block 181 represents the allocation of all blocks for all snapshots and the active file system in one range of 1,024 blocks. Each of the binary numbers 181, 182, 183, 184, and 190 in the array are a fixed length. In a preferred embodiment, the binary numbers are 16 bit numbers, although only 10 bits are used.

In a preferred embodiment, the large spacemap array binary number 182 (0000001111111110=1,021 in decimal units) tells the file system that the corresponding range is relatively full. In such embodiments, the largest binary number 00001111111111

(1,023 in decimal) represents a range containing at most one empty.. The small binary number 184 (0000000000001110=13 in decimal units) instructs the file system that the related range is relatively empty. The spacemap 180 is thus a representation in a very compact form of the allocation of all the blocks in the volume broken into 1,024 block sections. Each 16 bit number in the array of the spacemap 180 corresponds to the allocations of blocks in the range containing 1,024 blocks or about 4 MB. Each spacemap block 180 has about 2,000 binary numbers in the array and they describe the allocation status for 8 GB. Unlike the summary map 120, the spacemap block 180 needs to be determined whenever a file needs to be written.

Figure 2 shows a block diagram of an instant snapshot.

The old root block 200 of snapshot #1 201 includes the inode of the inode file 202 plus other information regarding the previous active file system known as snapshot #1 201, the snapmap 205, earlier active file systems known as snapshot #2 210, snapshot #3 215 and snapshot #4 220, and their respective snapmaps 225, 230 and 235.

The snapmap 205 of the previous active file system, snapshot #1 201, is a bitmap associated with the vacancy of blocks for snapshot #1 201. The respective snapmaps 225, 230 and 235 are earlier active maps that can be associated with particular snapshots 210, 215 and 220 and an inclusive OR summary map 245 of the snapmaps 225, 230 and 235. Also shown are other blocks 211 including double indirect blocks 240 and 332, indirect blocks 250, 251 and 252 and data blocks 260, 262, 263 and 264. Finally, Figure 2 shows the spacemap 270 of snapshot #1 201 including a collection of spacemap blocks of binary numbers 272, 273, 274, 275 and 276.

The old root block 200 includes a collection of pointers that are written to the previous active file system when the system had reached the previous CP. The pointers are aimed at a set of indirect (or triple indirect, or double indirect) inode blocks (not shown) or directly to the inode file 202 consisting of a set of blocks known as inode blocks 281, 282, 283, 284 and 285.

An inode block 281 in the inode file 202 points to other blocks 328 in the old root block 201 starting with double indirect blocks 240 and 332 (there could also be triple

indirect blocks). The double indirect blocks 240 and 332 include pointers to indirect blocks 250, 251 and 252. The indirect blocks 250, 251 and 252 include pointers that are directed to data leaf blocks 260, 262, 263 and 264 of the active file system 201.

Inode block 283 in the inode file 202 points to a set of blocks (1, 2, 3, ..., P) called the snapmap 205. Each block in the snapmap 205 is a bitmap where each bit corresponds to a block in the entire volume. A "1" in a particular position in the bitmap correlates with a particular allocated block in the active file system 201. Conversely, a "0" correlates to the particular block being free for allocation in the old root block 201. Each block in the snapmap 205 can describe up to 32K blocks or 128 MB.

Inode file 202 also includes inode block N 285. This block includes a set of pointers to a collection of earlier snapshots, snapshot #2 210, shapshot #3 215 and snapshot #4 220 of the volume. Each snapshot includes all the information of a root block and is equivalent to an older root block from a previous active file system.

Snapshot #1 201 also includes an old summary map 245 and old spacemap blocks 270. Although these blocks of data are included in snapshot #1 201 and previous snapshots, in a preferred embodiment, this data is not used by the active file system of figure 2.

Method of Use

Figure 3 shows a flow diagram of a method for using a system as shown in figure 1.

A method 300 is performed by the file system 100. Although the method 300 is described serially, the steps of the method 300 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 300 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 305, the file system 100 is ready to perform a method 300.

At a step 310, a user will request a snapshot of the file system 100.

At a step 315, a timer associated with the file system 100 initiates the creation of a new snapshot.

At a step 320, the file system 100 receives a request to make a snapshot.

At a step 325, the file system 100 creates a new file.

At a step 330, the root node of the new file points to the root node of the current active file system.

At a step 335, the file system 100 makes the file read only.

At a step 340, the file system 100 updates the new summary map by using an inclusive OR of the most recent snapmap and the existing summary file. This step must be done before any blocks are freed in the corresponding active map block. If multiple snapshots are created such that the processing overlaps in time, the update in step 340 need only be done for the most recently created snapshot.

At a flow point 345, the snapshot create and the summary file update is completed and the snapshot creation is done.

An analogous method may be performed for snapshot delete.

Figure 4 shows a flow diagram of a method for updating a summary map.

A method 400 is performed by the file system 100. Although the method 400 is described serially, the steps of the method 400 can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method 400 be performed in the same order in which this description lists the steps, except where so indicated.

At a flow point 410, the file system 100 is ready to update the summary map.

At a step 411, update of the summary map is triggered by a "snapdelete" command from an operator or user. As part of this step, the file system 100 receives and recognizes the "snapdelete" command.

At a step 412, the file system 100 responds immediately to the operator or user, and is ready to receive another operator or user command. However, while the operator or user sees a substantially immediate response, the file system 100 continues with the method 400 to process the "snapdelete" command.

At a step 413, the file system 100 marks an entry in the fsinfo block to show that the selected snapshot (designated by the "snapdelete" command) has been deleted.

At a step 414, the file system 100 examines the snapmap for the selected snapshot for blocks that were in use by the selected snapshot, but might now be eligible to be freed.

At a step 415, the file system 100 examines the snapmaps for (A) a snapshot just prior to the selected snapshot, and (B) a snapshot just after the selected snapshot. For blocks that were in use by the selected snapshot, the file system 100 sets the associated bit to indicate the block is FREE, only if both of those snapmaps show that the block was free for those snapshots as well.

The method 400 continues with the flow point 440.

At a step 421, update of the summary map is triggered by a write allocation operation by the file system 100. In a preferred embodiment, a write allocation operation occurs for a selected section of the mass storage. The "write allocation" operation refers to selection of free blocks to be seized and written to, as part of flushing data from a set of memory buffers to mass storage. As part of this step, the file system 100 determines a portion of the summary map corresponding to the selected section of the mass storage.

At a step 422, the file system 100 recalculates the summary map for the portion of the summary map corresponding to the selected section of the mass storage.

The method 400 continues with the flow point 440.

At a step 431, update of the summary map is triggered by a background operation. In a preferred embodiment, the file system 100 updates about one 4K data block of the summary map.

At a step 432, the file system 100 recalculates the summary map for the portion of the summary map selected to be updated.

The method 400 continues with the flow point 440.

At a flow point 440, the file system 110 has updated at least a portion of the summary map, and is ready to be triggered for further updates later.

Figure 5 shows a block diagram of copy-on-write maintenance of the active map.

When blocks are freed in the active map, the file system 110 is careful to not reuse those blocks until after a consistency point has passed (and thus that the newly free status of the block has been recorded in a snapshot). Accordingly, the file system 110 maintains two copies of the active map, a "true" copy 501 and a "safe" copy 502.

In normal operation 510 (outside a time when a consistency point is being generated), the file system 110 maintains both the "true" copy 501 and the "safe" copy 502 of the active map. Since in normal operation 510 blocks can only be freed, not allocated, only changes from IN-USE to FREE are allowed. The file system 110 makes all such changes in the "true" copy 501, but does not make them to the "safe" copy 502. The "safe" copy 502 therefore indicates those blocks which can be safely allocated at the next consistency point.

While generating a consistency point, during a write allocation interval 520, blocks can be either freed (by continued operation of the file system 110) or allocated (by the

write allocation operation). Both types of change are made to both the "true" copy 501 and the "safe" copy 502.

While still generating a consistency point, during a flush data to disk interval 530, blocks can again only be freed (by continued operation of the file system 110); they cannot be allocated because the write allocation interval 520 is finished for that consistency point. The file system 110 makes all such changes in the "safe" copy 502, but does not make them to the "true" copy 501. At the end of the flush data to disk interval 530, the file system 110 switches the roles of the "true" copy 501 and the "safe" copy 502, so that all such changes were in fact made to the new "true" copy 501 only.

Alternative Embodiments

Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

Claims

1. A method for capturing the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including steps for

recording an active map in said file system of said storage blocks not available for writing data;

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time; and

refraining from writing data to storage blocks in response to said active map, and at least one of said copy of said active map at said previous time.

2. A method as in claim 1, wherein said step for refraining includes determining a logical union of said storage blocks used by one or more of said copies of said active map at said previous time.

3. A method as in claim 1, wherein said step for refraining includes determining a subset of said storage blocks used by one or more of said copies of said active map at said previous time.

4. A method as in claim 1, wherein said file system is a WAFL file system.

5. A method as in claim 1, wherein said active map at said previous time is a snapmap.

6. A method as in claim 1 and 5, including removing a root inode of said snapmap using a snap delete.

7. A method as in claim 6, including steps for determining not to write to a block after said step, provided the previous or next snapmap uses said block.

8. A method as in claim 1, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block.

9. A method for capturing the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time; and

returning to said file system at a previous time using said consistent version of said file system following an unintended deletion or modification.

10. A method as in claim 9, wherein said consistent version includes a pointer to a previous root block of the inode file.

11. A method as in claim 9, wherein said file system is a WAFL file system.

12. A method as in claim 9, wherein said active map at said previous time is a snapmap.

13. A method as in claim 9 and 12, including a snapdelete method for removing a root inode of said snapmap.

14. A method as in claim 13, including steps for determining not to write to a block after said snapdelete method provided a previous or next snapmap uses said block.

15. A method as in claim 9, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block.

16. A method for saving previous versions of an active file system including the contents of the files and directories in a file system, said file system comprising a set of storage blocks in a mass storage system including steps for

writing modified files to unused data blocks;

keeping previous files in currently occupied blocks; and

recording a consistency point in said file system including a consistent version of said file system at a previous time, said consistency point including a copy of said active map at said previous time;

17. A method as in claim 16, including retrieving said file system at a previous time using a pointer.

18. A method as in claim 16, wherein said pointer corresponds to a root block of said file system at a previous time.

19. A method as in claim 16, wherein said file system is a WAFL file system.

20. A method as in claim 16, wherein said active map at said previous time is a snapmap.

21. A method as in claim 16 and 20, including a snapdelete method for removing a root inode of said snapmap.

22. A method as in claim 20, including not writing to a block after said snapdelete method provided a previous or next snapmap uses said block.

23. A method as in claim 16, including a copy-on-write mechanism for copying modified data to a new block and saving old data in a current data block..

24. A method of operating a file system, said file system including an active map of information indicating in-use and free blocks, said file system maintaining a set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said method including

making write allocation decisions in response to a copy of an earlier active map included in at least one of said snapshots.

25. A method of operating a file system, said file system including an active map of information indicating in-use and free blocks, said file system maintaining a

set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said method including

computing a summary map in response to at least one copy of an earlier active map included in at least one of said snapshots.

26. A method as in claim 25, including

making write allocation decisions in response to said summary map.

27. A method as in claim 25, wherein

said set of snapshots includes at least two said snapshots; and

a result of said computing includes an indicator of a union of all blocks indicated by at least two said copies of earlier active maps included in said set of snapshots.

28. A method as in claim 25, wherein

said set of snapshots includes at least two said snapshots; and

said computing includes performing a bitwise logical operation on at least two said copies of earlier active maps included in said set of snapshots.

29. A method as in claim 25, including

making write allocation decisions both in response to a current active map and in response to said summary map.

30. A method as in claim 25, including

computing a combination of a current active map and said summary map; and

making write allocation decisions in response to a result of said computing.

31. A method as in claim 25, including, for a selected portion of said summary map

identifying a set of snapshots created since a recent update of said selected portion; and

updating said selected portion in response to only a most recent one of said snapshots.

32. In a file system including an active map of information indicating in-use and free blocks, said file system maintaining a set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said file system maintaining a summary map in response to at least one copy of an earlier active map included in at least one of said snapshots, a method of updating said summary map, said method including

receiving a request to delete a selected snapshot;

for a block used by said selected snapshot, indicating said block is free in said summary map only in response to a snapshot just prior to said selected snapshot and in response to a snapshot just after said selected snapshot.

33. A method as in claim 32, wherein said indicating frees said block only when both

said block is unused by said snapshot just prior to said selected snapshot; and

said block is unused by said snapshot just after said selected snapshot.

34. A method as in claim 32, wherein said snapshot just after said selected snapshot corresponds to an active file system.

35. In a file system including an active map of information indicating in-use and free blocks, said file system maintaining a set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said file system maintaining a summary map in response to at least one copy of an earlier active map included in at least one of said snapshots, a method of updating said summary map, said method including

selecting a set of blocks maintained by said file system for which to perform a write allocation operation;

updating only a portion of said summary map corresponding to said set of blocks, in response to said selecting; and

performing said write allocation operation in response to said updated summary map.

36. In a file system including an active map of information indicating in-use and free blocks, and said file system maintaining a set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said file system maintaining a summary map in response to at least one copy of an earlier active map

included in at least one of said snapshots, a method of updating said summary map, said method including

while generating a consistency point, selecting a set of blocks maintained by said file system and updating only a portion of said summary map corresponding to said set of blocks.

37. In a file system including an active map of information indicating in-use and free blocks, and said file system maintaining a set of snapshots, each snapshot including a representation of said file system as it was at an earlier time, said file system maintaining a summary map in response to at least one copy of an earlier active map included in at least one of said snapshots, a method of updating said summary map, said method including

refraining from indicating a selected block as free in response to whether said selected block is included in said copy of an earlier active map.

38. In a file system including an active map of information indicating in-use and free blocks, a method of updating said active map, said method including

maintaining a plurality of copies of said active map, at least a first said copy being a substantially true representation of in-use and free blocks, and at least a second said copy being a representation of in-use and free blocks which reflects fewer free blocks than said first copy; and

wherein said second copy refrains from indicating a selected block as free until after a next consistency point is completed.

39. A method as in claim 38, including

swapping said second copy with said first copy after said consistency point is completed.

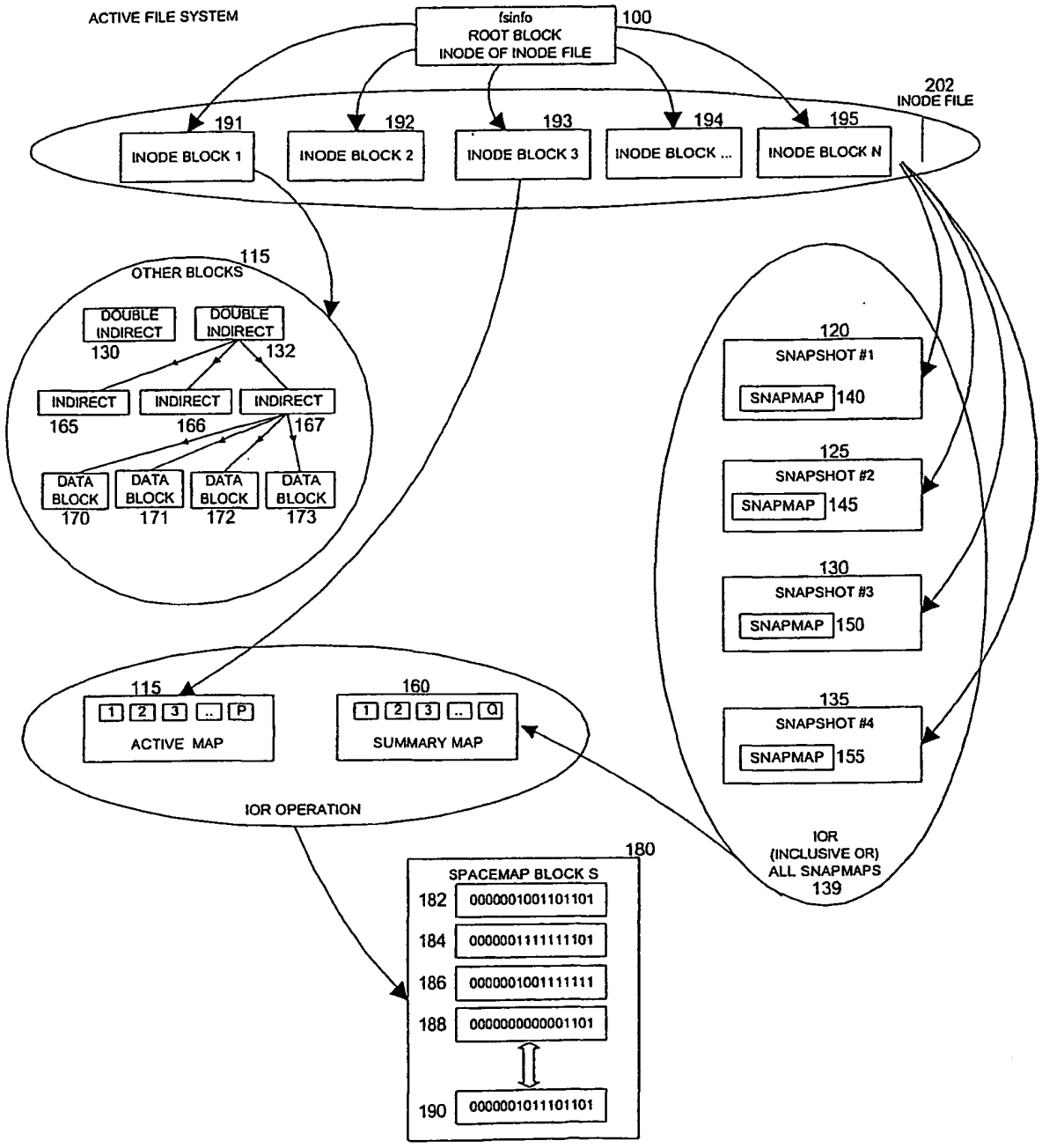

ACTIVE FILE SYSTEM
fsinfo
ROOT BLOCK
INODE OF INODE FILE
100
202
INODE FILE
191
192
193
194
195
INODE BLOCK 1
INODE BLOCK 2
INODE BLOCK 3
INODE BLOCK ...
INODE BLOCK N
115
OTHER BLOCKS
DOUBLE INDIRECT
130
DOUBLE INDIRECT
132
INDIRECT
165
INDIRECT
166
INDIRECT
167
DATA BLOCK
170
DATA BLOCK
171
DATA BLOCK
172
DATA BLOCK
173
120
SNAPSHOT #1
SNAPMAP 140
125
SNAPSHOT #2
SNAPMAP 145
130
SNAPSHOT #3
SNAPMAP 150
135
SNAPSHOT #4
SNAPMAP 155
115
1 2 3 .. P
ACTIVE MAP
160
1 2 3 .. Q
SUMMARY MAP
IOR OPERATION
IOR
(INCLUSIVE OR)
ALL SNAPMAPS
139
180
SPACEMAP BLOCK S
182 0000001001101101
184 0000001111111101
186 0000001001111111
188 0000000000001101
190 0000001011101101

Fig. 1

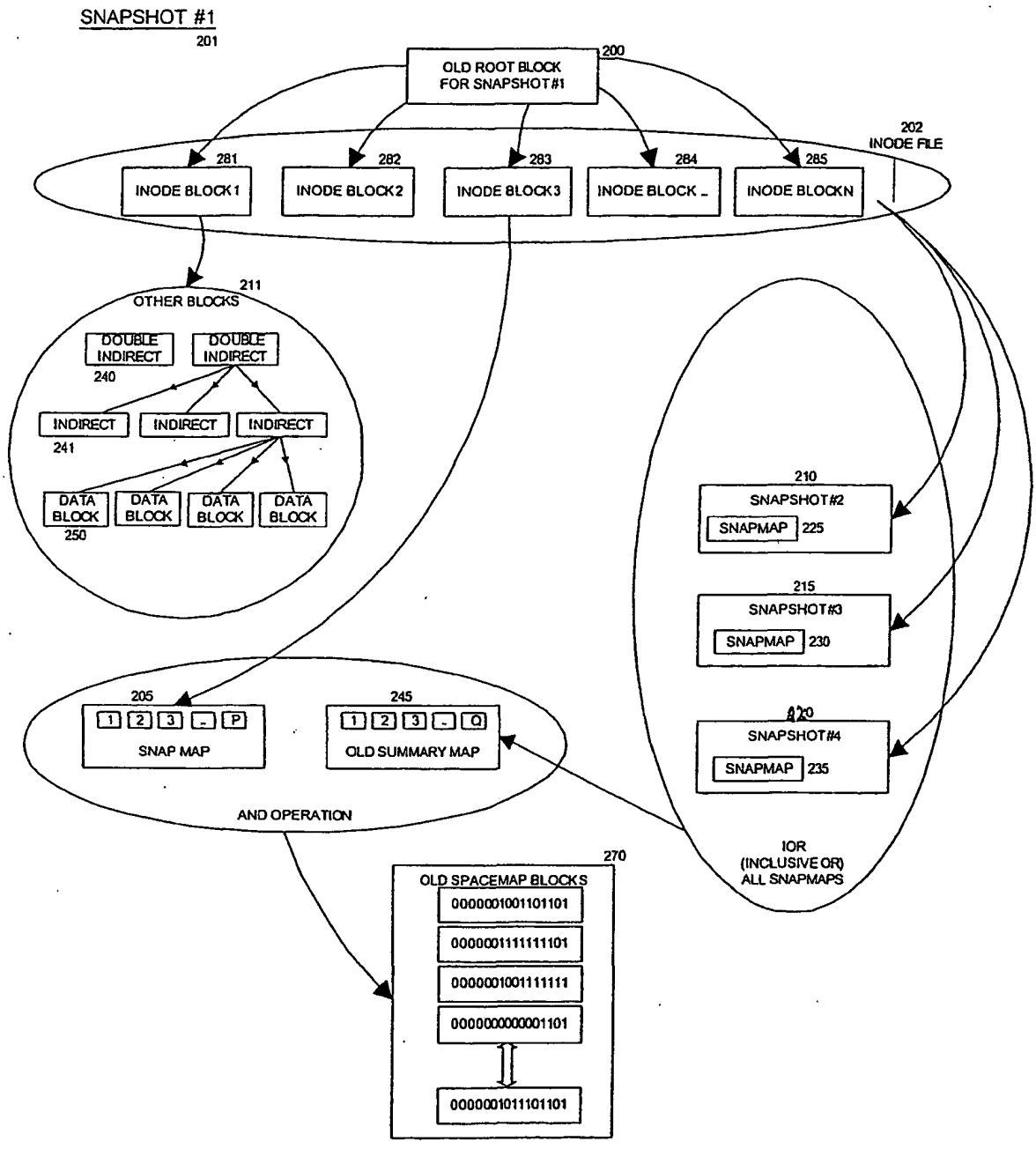

SNAPSHOT #1
201
OLD ROOT BLOCK FOR SNAPSHOT#1
200
202
INODE FILE
281
282
283
284
285
INODE BLOCK1
INODE BLOCK2
INODE BLOCK3
INODE BLOCK _
INODE BLOCKN
211
OTHER BLOCKS
DOUBLE INDIRECT
DOUBLE INDIRECT
240
INDIRECT
INDIRECT
INDIRECT
241
DATA BLOCK
DATA BLOCK
DATA BLOCK
DATA BLOCK
250
210
SNAPSHOT#2
SNAPMAP 225
215
SNAPSHOT#3
SNAPMAP 230
220
SNAPSHOT#4
SNAPMAP 235
205
1 2 3 _ P
SNAP MAP
245
1 2 3 _ Q
OLD SUMMARY MAP
AND OPERATION
IOR (INCLUSIVE OR) ALL SNAPMAPS
270
OLD SPACEMAP BLOCKS
0000001001101101
0000001111111101
0000001001111111
0000000000001101
0000001011101101

Fig. 2

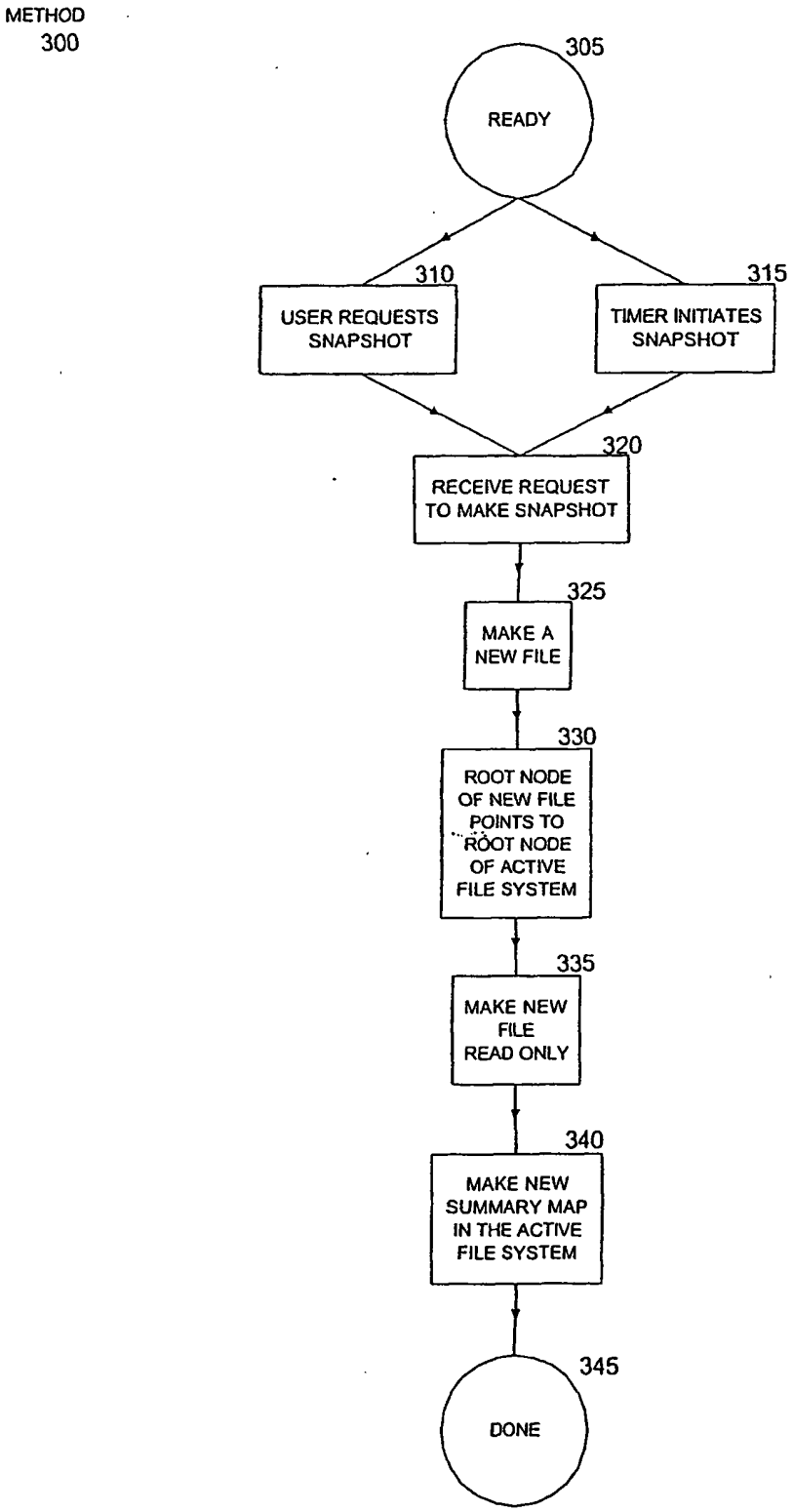

METHOD
300
305
READY
310
USER REQUESTS SNAPSHOT
315
TIMER INITIATES SNAPSHOT
320
RECEIVE REQUEST TO MAKE SNAPSHOT
325
MAKE A NEW FILE
330
ROOT NODE OF NEW FILE POINTS TO ROOT NODE OF ACTIVE FILE SYSTEM
335
MAKE NEW FILE READ ONLY
340
MAKE NEW SUMMARY MAP IN THE ACTIVE FILE SYSTEM
345
DONE

Fig. 3

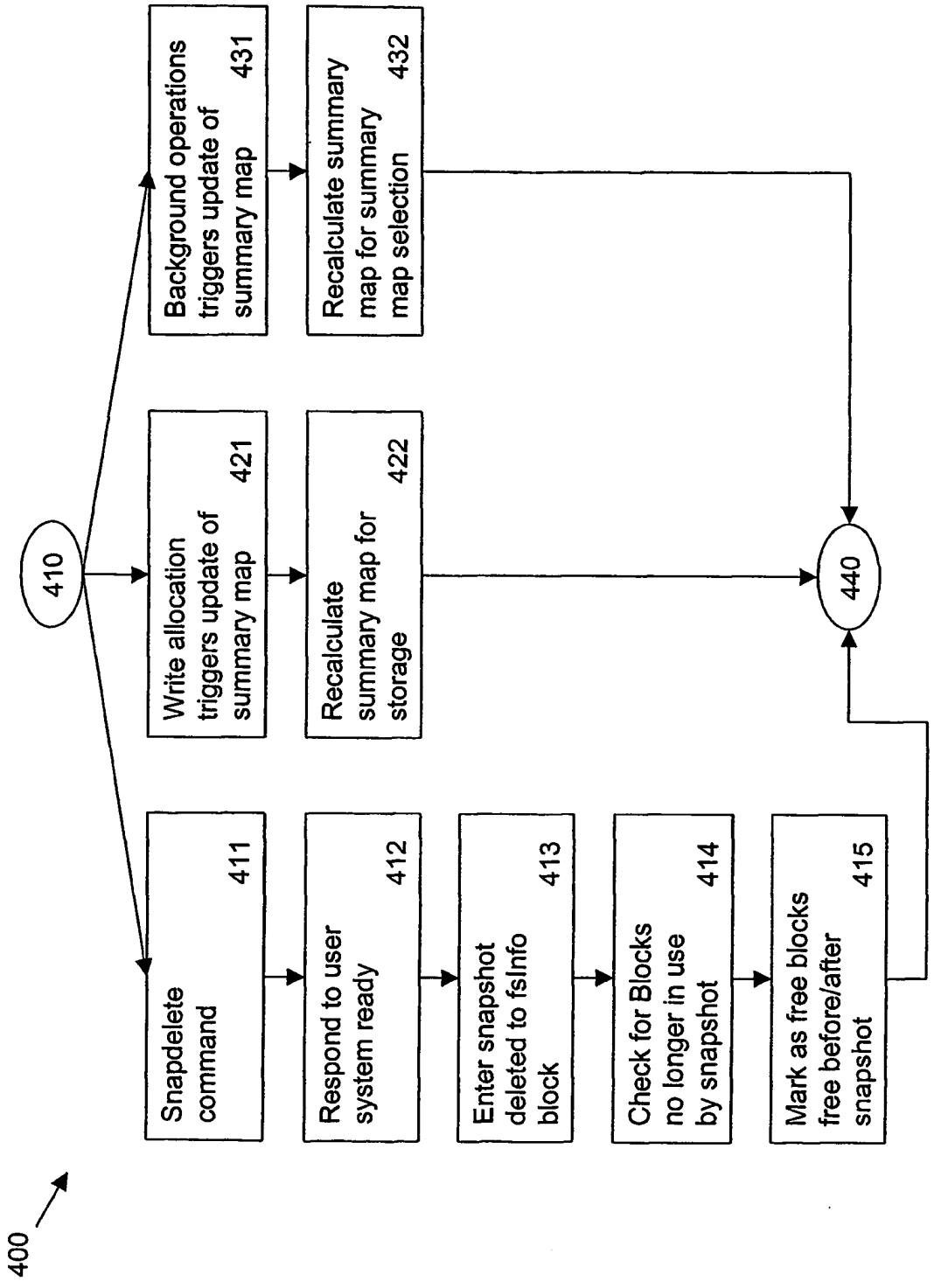

400
410
Snapdelete command 411
Respond to user system ready 412
Enter snapshot deleted to fsInfo block 413
Check for Blocks no longer in use by snapshot 414
Mark as free blocks free before/after snapshot 415
Write allocation triggers update of summary map 421
Recalculate summary map for storage 422
Background operations triggers update of summary map 431
Recalculate summary map for summary map selection 432
440

Fig. 4

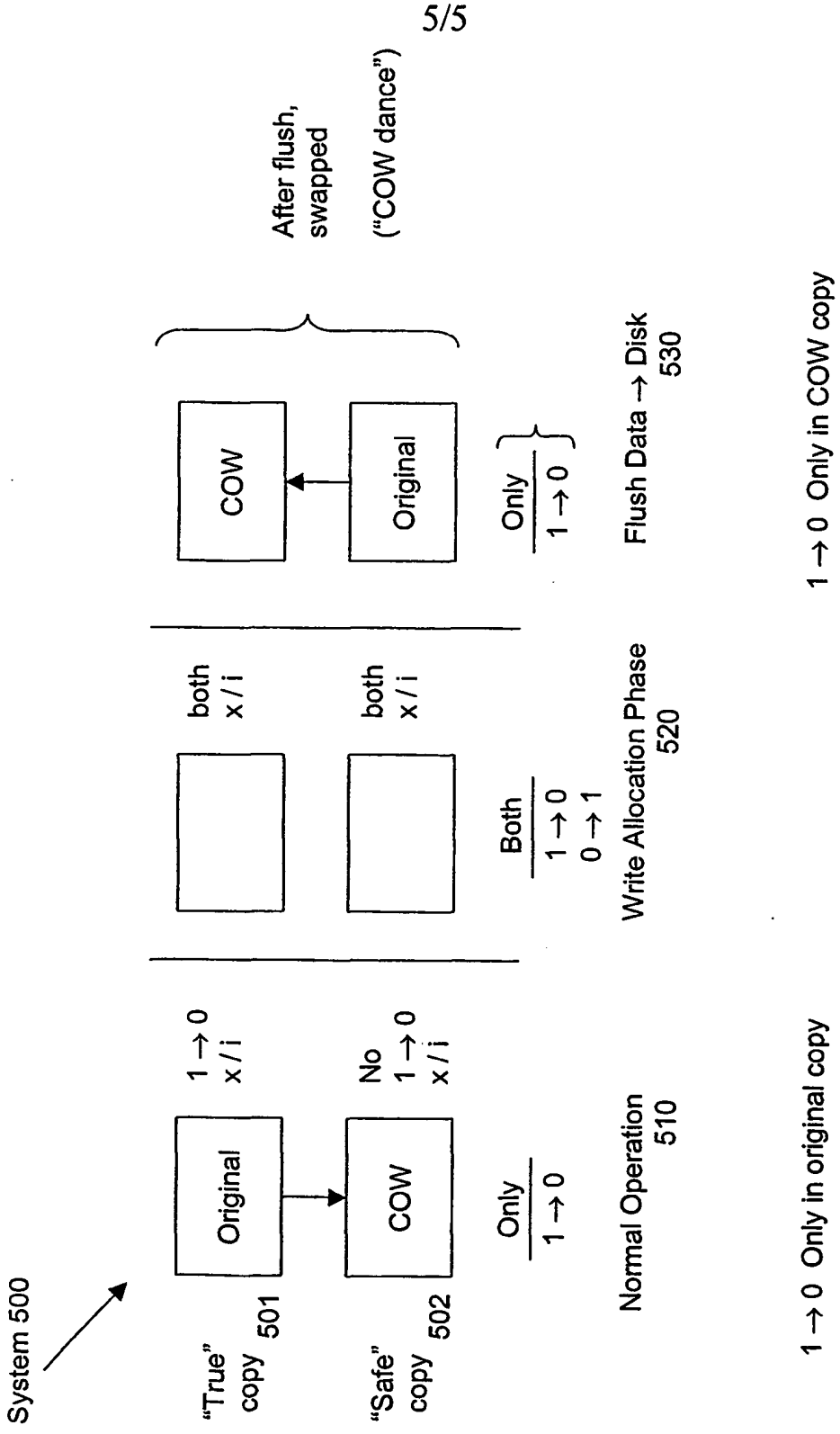

System 500
"True" copy 501
"Safe" copy 502
Original
COW
1 → 0 x / i
No 1 → 0 x / i
Only 1 → 0
Normal Operation 510
1 → 0 Only in original copy
both x / i
both x / i
Both 1 → 0 0 → 1
Write Allocation Phase 520
COW
Original
Only 1 → 0
Flush Data → Disk 530
1 → 0 Only in COW copy
After flush, swapped ("COW dance")

Fig. 5.